Comparing a Multilayer Perceptron with Convolutional Neural Networks for Facial Emotion Recognition

Benedetta Bruno, Vaitian L. Marianayagam, Cecilie S. Rønnestad & Sander S. Svartbekk
University of Oslo, Department of Physics

(Dated: December 2025)

Facial emotion recognition is a topic of growing interest in both industry and basic research. We compare a multilayer perceptron (MLP) model with convolutional neural networks (CNNs) in the performance of this task. We try a CNN architecture with and without Squeeze-and-Excitation (SE) blocks, which have improved the accuracy of CNNs in computer vision in other studies. The models are trained on the Real-World Affective Faces Database, containing 15 339 labeled images of emoting faces. We observe the best results for the CNN without SE blocks, with 76.60 % test accuracy. The CNN with SE blocks achieves a test accuracy of 74.22 %, and the MLP achieves 68.87 %. All three models outperformed the larger SE-ResNet used by Huang *et al.*, which achieved 65.67 %. Our results are not conclusive on whether SE blocks improve FER performance in the tested CNNs. The performances highlights that well optimized smaller networks can outperform larger networks with less care taken into hyperparameter tuning and architectural choices. The CNNs performed substantially better than the MLP, supporting standard practice in the field.

I. INTRODUCTION

Facial emotion recognition (FER) is a category of computer vision where the program is given a picture of a human face and is tasked with interpreting the expressed emotion. The field has attracted growing interest across different industries, with use cases including analysis of viewer-reaction to ads, adjusting safety systems in cars when drivers are showing signs of tiredness or stress, and pain detection in healthcare [1]. It is also an interesting topic in psychology and neuroscience research, as the models used for FER can indicate which parts of the face contains most information about a person's emotions [2].

At the same time, the use of FER raises important ethical questions. Emotional responses are highly sensitive data, and large-scale FER systems are easy to deploy without explicit consent, for example in online platforms and public spaces. There are also privacy concerns related to the storage and reuse of facial images, as well as the potential for misuse in surveillance or image manipulation. Additionally, if training datasets are not representative of the population, FER models might have systematic biases across age, gender, or ethnicity, which can reinforce existing social inequalities.

From a technical perspective, FER is a challenging problem, even without considering ethical issues. Human annotators do not always agree on which emotion a face expresses, and the underlying emotional state may be mixed or ambiguous rather then belonging to a single discrete class. This makes both label noise and overlap between classes an inherent problem. Real-world images also vary in pose, lighting, and resolution, so reliable models should be able to sort out the discriminating image features in a variety of conditions. An imbalance in the classes is also common, with some emotions being much more frequent than others. This may vary between image sources, for example people tend to be smiling in staged photos, while neutral facial expressions will be more common in surveillance video. This may introduce a bias in models towards the majority classes in the training data.

In this project, we compare three architectures on the task of FER. A baseline Multilayer Perceptron (MLP), a Convolutional Neural Network (CNN), and a CNN augmented with Squeeze-Excitation (SE) blocks. The three models are comparative in size with regards to trainable parameters. They are trained and evaluated on the Real-World Affective Faces Database (RAF-DB) [3], which contains facial images labeled with either of seven basic emotions. We compare our final results with Huang *et al.* [2], who also trained a CNN on the RAF-DB.

A detailed description of the dataset and the models used, along with a theoretical explanation of the methods, are found in Section II. Results and analysis are presented in Section III, while the main findings and conclusions are summarized in Section IV. Instructions for accessing the dataset to reproduce our results are given in Section V, and all code related to this project is available in a public GitHub repository[1].

II. METHODS

A. The Dataset

RAF-DB contains 15 339 images of faces, each labeled with one of seven emotions: surprise, fear, disgust, happiness, sadness, anger or neutral. There is a large diversity in the dataset of head poses, ages, facial occlusions, lighting and image resolutions. There are multiple racial groups represented in the images, but a majority are reported as "Caucasian" (77 %) by the authors [3]. 15% of the imaged people are reported as "Asian", and 8%

[1] `https://github.com/Cecils0505/facial-emotion-recognition`

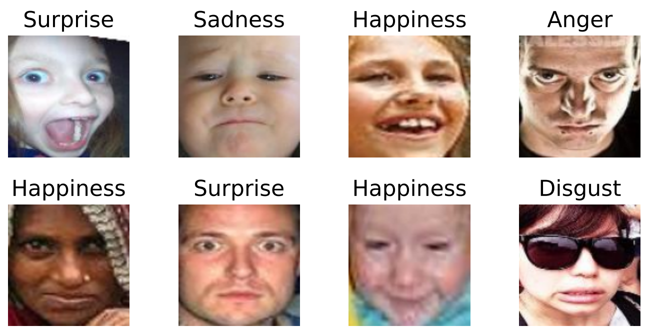


Figure 1. **Examples from the dataset.** Eight images from the dataset, showing a variety of ages, facial occlusions, ethnicities, lighting and resolutions. Note that all images are 100×100 pixels, regardless of apparent resolution.

as "African-American". All images have been resized to 100×100 pixels, and are centered using the eyes and center of the mouth. Examples of the images are shown in Figure 1. The dataset comes with a standard division of 12 271 training samples and 3 068 test samples, meaning that the same train-test split will be used between different studies. This eliminates some noise when doing comparisons. The train and test sets have similar distributions of emotion labels, as seen in Table I. There is, however, large variation in sample size between the different labels, with the largest discrepancy being between Happiness (38.89 % in train) and Fear (2.29 % in train), with more than an order of magnitude difference.

Table I. **Distribution of labels.** Percentage of emotion labels in train and test sets of the dataset.

Label	Train [%]	Test [%]
Surprise	10.51	10.72
Fear	2.29	2.41
Disgust	5.84	5.22
Happiness	38.89	38.62
Sadness	16.15	15.68
Anger	5.75	5.28
Neutral	20.57	22.16

1. Transformations

Two transformations are applied to the images to make them more digestible to the neural networks. First, the RGB values v are compressed from $v \in [0,\, 255]$ to $\tilde{v}^{(1)} \in [0,\, 1]$ by the simple transformation

$$\tilde{v}^{(1)} = v/255. \tag{1}$$

The values are then stretched to $\tilde{v}^{(2)} \in [-1,\, 1]$ by

$$\tilde{v}^{(2)} = \frac{\tilde{v}^{(1)} - 0.5}{0.5} = 2\tilde{v}^{(1)} - 1, \tag{2}$$

which makes them approximately zero-centered. The transformed values $\tilde{v}^{(2)}$ are used as input for the networks.

B. Multilayer Perceptrons

An MLP is organized into layers of nodes, or neurons, which each applies a non-linear transformation on their input by applying a weight, a bias and an activation function. The layers are "fully connected", meaning that all nodes of each layer are connected to all nodes of adjacent layers. A basic MLP architecture is shown in Figure 2. The weights $\boldsymbol{W}^{(l)}$ and biases $\boldsymbol{b}^{(l)}$ of layer l are applied to

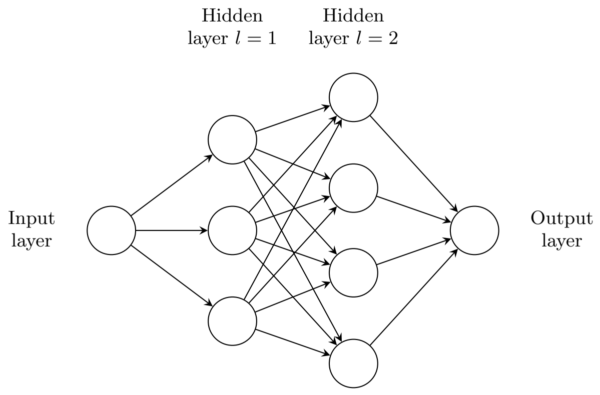


Figure 2. **Schematic of two–layer MLP.** The input and output layers have one neuron each. The first hidden layer has 3 neurons. The second hidden layer has 4 neurons. Each neuron in the source layer is connected to every neuron in the destination layer – the network is fully connected.

the output values $\boldsymbol{a}^{l-1}$ of the previous layer by

$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)}. \tag{3}$$

Next, the non-linear activation function $\boldsymbol{f}^{(l)}$ is applied before passing the values on to the next layer,

$$\boldsymbol{a}^{(l)} = \boldsymbol{f}^{(l)}\big(\boldsymbol{z}^{(l)}\big). \tag{4}$$

The output of the last layer $\boldsymbol{a}^{(L)}$ is the prediction of the model. The number of nodes in this layer must therefore correspond to the number of classes in the problem. In our setting, the output layer consists of seven nodes, corresponding to the seven emotion classes. Similarly, the number of nodes in the first layer must correspond to the number of values in the input. The whole forward pass can be summarized compactly as

$$\begin{aligned}\boldsymbol{a}^{(L)} = \boldsymbol{f}^{(L)}\Big(\boldsymbol{W}^{(L)}\boldsymbol{f}^{(L-1)}\Big(\dots\boldsymbol{f}^{(1)}\big(\boldsymbol{W}^{(1)}\boldsymbol{a}^{(0)} + \boldsymbol{b}^{(1)}\big)\\ + \boldsymbol{b}^{(2)}\dots\Big) + \boldsymbol{b}^{(L)}\Big)\end{aligned} \tag{5}$$

where $\boldsymbol{a}^{(0)}$ is the input data and $\boldsymbol{a}^{(L)}$ is the output.

To evaluate the quality of the MLP's predictions, a cost function is applied to the output. The MLPs are optimized by gradient descent methods, where the weights and biases are updated according to the gradient of the cost function. This can be iteratively calculated through a method known as backpropagation:

$$\frac{\partial C}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} a_j^{(l-1)}, \qquad \frac{\partial C}{\partial b_i^{(l)}} = \delta_i^{(l)}, \tag{6}$$

where C is the cost function, w_{ij} is the weight between node i in one layer, and node j in the previous, $a_j^{(l-1)}$ is the output from node j, b_i is the bias of node i and δ_i is the error propagated through the layers to node i. δ_i is calculated iteratively by

$$\delta_i^{(l)} = \sum_j \delta_j^{(l+1)} w_{ji}^{(l+1)} f'^{(l)}\left(z_i^{(l)}\right), \tag{7}$$

where $f'^{(l)}\left(z_i^{(l)}\right)$ is the derivative of the activation at node i. This calculation is done from the output layer, back towards the input layer. When the gradient is calculated, an update algorithm can be used to improve the weights and biases. The update algorithm is usually a first order approximation of the Newton–Raphson method. For a more thorough discussion of backpropagation, see Bruno *et al.* [4].

C. Convolutional Neural Networks

MLPs and CNNs are two different types of Feedforward Neural Networks, as information only flows in one direction in both architectures. The key difference is that MLPs only contain fully connected neural layers, whereas CNNs also contain convolutional layers. The transformations done by the fully connected layers are affine transformations, and do not necessarily exploit specific properties of the input; topological information is not taken into account, as all axes are treated equally. The data might thus have implicit structure that is lost in an MLP. Discrete convolutions, on the other hand, preserve some types of internal structure of the data.

Unlike regular neural network layers, the convolutional layers have weights arranged in tensors, which can have the same dimensions as images: width W, height H, and depth D. Width and height correspond to the size of the image (number of pixels along the sides), and the depth corresponds to color channels; three for color images, and one for black and white. In addition to convolutional and fully connected layers, CNNs typically also contain pooling layers. Their function is to reduce the dimensionality of the convolution outputs, without removing much of the information from the convolutions. Both convolutional and fully connected layers typically include an activation function pass before the values are sent to the next layer.

Convolutional Layer: Takes in a volume and slides filters (kernels) across it. At every spatial location, it computes a dot product between the overlapping filter weights and input subregion, making a feature map.

Pooling Layer: Takes subregions of the input data and applies a function – typically either the max value or the mean value in the given "patch". It then passes on this new, shrunk, collection of values, while preserving the overall structure of the data.

Fully Connected Layer: Contains a weight and a bias, and is connected to all the nodes in the previous layer. The volume, that is the output of the previous layer, is flattened, and the fully connected layer will then compute the class scores.

In a typical CNN architecture different types of layers are stacked. CNNs transform the input image layer by layer from the original pixel values to the output class scores. The output layer is typically a fully connected layer.

The main principles that motivate the use of convolutions are locality of information and repetition of patterns within a signal. Images usually exhibit strong local correlations, meaning that a small region of the image tends to be similar to its neighboring areas. Therefore, rather than linking every pixel to a neuron in the first hidden layer, as is done in MLPs, each neuron is instead connected to only a small region of the image (across all three RGB channels). The size of this small region is fixed and is referred to as a receptive field.

1. *Convolutions and pooling*

A convolution is a mathematical operation on two functions f and g that produces a third function $f * g$, as the integral of the product between the two functions after one is reflected about the y-axis and shifted [5]. A convolution is defined as

$$y(t) = \int_{-\infty}^{\infty} x(a) w(t-a) da = (x * w)(t). \tag{8}$$

Here, $x(a)$ is the input and $w(t-a)$ is the so-called weight function or kernel. $y(t)$ is a continuous function. The discretised version of Equation (8) is

$$y(t) = \sum_{a=-\infty}^{\infty} x(a) w(t-a). \tag{9}$$

Convolution is a commutative operation. Graphically, it expresses how the shape of one function is modified by the other. It is often useful to implement convolutions on more than one dimension at a time. With a two dimensional image I as input and a filter defined by a two dimensional kernel K, we can produce the output S as

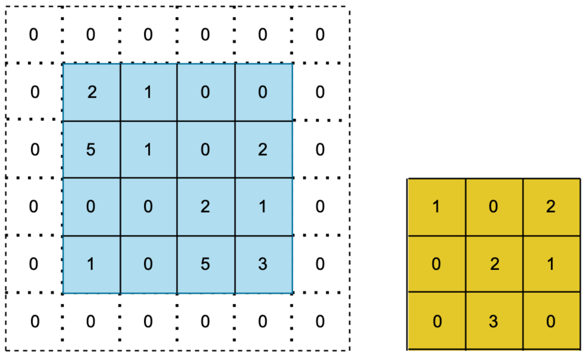
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n) K(i-m, j-n). \tag{10}$$

Due to commutative properties, Equation (10) can be rewritten as

$$S(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n), \tag{11}$$

which is more straightforward to implement in a machine learning library.

The object used as input (for example a digital image) is called an input feature matrix. The kernel "slides" across the input feature matrix at a rate determined by the so-called stride. The stride s is essentially the distance between two consecutive positions of the kernel. The area of the input matrix overlapping with the kernel is convolved with the kernel itself. Each convolution produces a value on the output feature matrix. Zero padding – zeros concatenated at key locations of the input matrix – helps us control the size of the output matrix. Figure 3 shows an example input feature matrix (in blue) with zero padding along both axes, and an example kernel (in yellow). Convolving the two with stride $s = 1$ produces the output feature matrix in Figure 4. The procedure



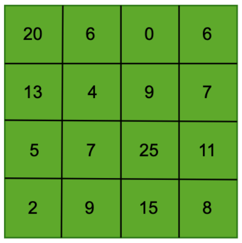
0	0	0	0	0	0
0	2	1	0	0	0
0	5	1	0	2	0
0	0	0	2	1	0
0	1	0	5	3	0
0	0	0	0	0	0

1	0	2
0	2	1
0	3	0

Figure 3. **Padded input feature matrix** (left) **and kernel** (right). The kernel slides across the input feature matrix and the overlapping regions are convolved at each step, with step size defined by the stride. The zero padding along both axes ensures that the output size is the same as the input size. This is sometimes referred to as *same* padding [6].

can be repeated using different kernels to form as many ("intermediate") feature maps as desired. These are then summed up elementwise to form the output feature map. In the case of image processing, we essentially have three stacked 2D feature matrices – one for each color (R,G,B), forming the input image.

Convolutional layer properties do not interact across axes, meaning that the choice of kernel size, stride, and



20	6	0	6
13	4	9	7
5	7	25	11
2	9	15	8

Figure 4. **Output feature matrix.** Obtained from the convolution of input feature matrix with kernel in Figure 3, with stride $s = 1$ along both axes.

zero padding along axis i only affects the the output size of axis i [6]. The output size is equal to the amount of positions the kernel can occupy across the input matrix. We start with input size W. Padding adds P pixels to one end of an axis and P pixels to the other end, so the effective width is $W + 2P$. A kernel of size K takes K pixels to fit. One kernel position counts as one output value. The amount of available positions is $(W + 2P - K)/s$, where s is the stride. The output size W_{out} is therefore given by

$$W_{\text{out}} = \left\lfloor \frac{W - K + 2P}{s} \right\rfloor + 1, \tag{12}$$

where we account for the starting position of the kernel by adding 1.

Pooling layers constitute another important building block of CNNs. Pooling operations reduce the size of input feature maps by applying a function – typically averaging or finding the maximum value – to subregions. Pooling works very much like a discrete convolution, but replaces the actual operation of convolution at each site with a pooling function. As such, the output sizes for pooling is also given by Equation (12).

The number of weights of a convolutional layer is determined by the number of output channels (filters) C_{out}, the number of input filters C_{in}, and the kernel size K as $(C_{\text{out}} \times C_{\text{in}} \times K^2)$. Additionally, each filter has one bias, giving C_{out} biases. The total amount of trainable parameters per convolutional layer is then

$$C_{\text{out}}(C_{\text{in}} K^2 + 1). \tag{13}$$

Both stride and padding are treated as hyperparameters. Pooling and activation layers typically do not have trainable parameters, so they are all contained in convolutional and fully connected layers. Like MLPs, CNNs are trained with backpropagation.

D. Squeeze-and-Excitation blocks

Squeeze-and-Excitation (SE) blocks are small modules that can be added on top of ordinary convolutional layers.

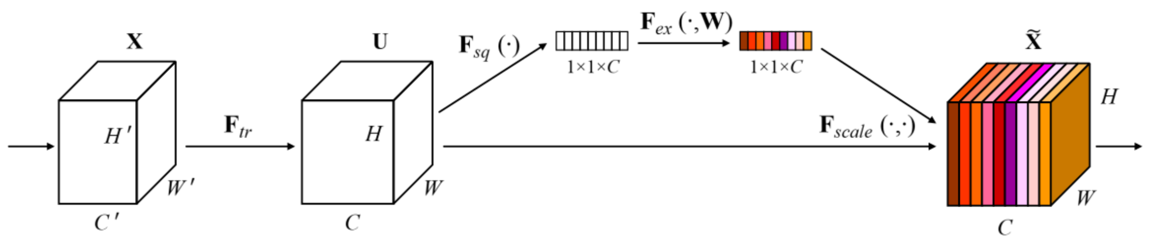


Figure 5. **Schematic of a Squeeze-and-Excitation block.** The feature maps U produced by a convolutional layer are first globally pooled ("squeeze") to obtain a channel descriptor, then passed through a small gating network ("excitation"), that outputs channel-wise weights. These are used to rescale the original feature maps, amplifying informative channels and suppressing less useful ones. Reproduced from Hu *et al.* [7]

They let the network evaluate which feature channels are most important for a given input. Instead of treating all feature channels equally, the network learns to enhance informative channels, and suppress less useful ones, based on the current input. [7, 8]

An SE block works in three steps. First, the squeeze step. Here, the output of a convolutional layer (a feature map of size $H \times W \times C$) is compressed into a vector of length C, by taking the average over all spatial positions in each channel. This gives one number per channel, that summarizes how active that channel is across the whole image. Next is the excitation step. The C-dimensional vector is passed through a small two-layer fully connected network with a bottleneck. It first reduces the dimension, applies a ReLU function, and expands back to C, and finally applies a sigmoid. This produces C weights between 0 and 1, one for each channel, that encodes how important that channel is for the current input. In the last step, these weights are used to rescale the original channel maps feature-wise, so the channels deemed important are amplified, and less useful ones are down-weighed. A schematic overview of an SE block, is shown in Figure 5. The output has the same shape as the input feature and comparatively few trainable parameters, so SE blocks can be inserted into existing CNN architectures with minimal changes to the overall design. [7].

E. Dropout

We are fitting models with millions of parameters to a dataset containing tens of thousands of samples. Because of this large difference in size between model and dataset, some degree of overfitting is expected. We employ the regularization technique known as Dropout [9] to mitigate this.

One method of avoiding overfitting is to train several different models on the same data, which would then model (and possibly overfit to) the data in different ways. One could then average over predictions from all models, and thus extract one prediction from the combined model. For large models, this would have a very high computational cost. Dropout is a way of approximating this strategy. Before each parameter update cycle, a random subset of the nodes are deactivated, each with probability $1 - p$, such that p is the probability of a node to be active. The retained nodes are then updated once, as if they constituted the whole model. Before the next cycle, all nodes are turned on again, and a new random subset is turned off. This is repeated throughout training. Before predictions are made, all weights are multiplied by p. This has the effect of approximating the geometric mean of 2^n models with $n/2$ nodes each.

An important consideration is that dropout reduces the effective size of the network. At any time during training, only pn nodes are present, and they are not able to make co-adaptions with the other $(1-p)n$ nodes. As such, if one expects n nodes to be optimal for a given task, at least n/p nodes should be used if Dropout is employed.

Table II. **MLP architecture.** The layers are listed in order, so the output size of the layer above will be the input size of the layer below. The first input size is $3 \times 100 \times 100$, corresponding to three color channels and 100×100 pixels in each image.

Layer	Output size	Trainable parameters
Fully connected	132	3 960 132
Fully connected	2048	272 384
Output	7	14 343
Total	7	4 246 859

F. Models and training scheme

We are training three models to recognize facial emotional expressions; a two–layer MLP, a CNN with SE-blocks, and a CNN without. We will refer to the CNN architecture with the SE blocks as "SE-CNN", and the one without as "plain CNN". The architectures are outlined in Tables II, III and IV.

All convolutional and fully connected layers apply

Table III. **Plain CNN architecture.** The layers are listed in order, so the output size of the layer above will be the input size of the layer below. The first input size is $3 \times 100 \times 100$, corresponding to three color channels and 100×100 pixels in each image. The brackets in the convolutional (conv) and pooling layers contain [kernel/stride/padding].

Layer	Output size	Trainable parameters
Conv [10/3/4]	$32 \times 33 \times 33$	9 632
Conv [3/1/1]	$64 \times 33 \times 33$	18 496
Avg pooling [2/2/0]	$64 \times 16 \times 16$	0
Fully connected	256	4 194 560
Output	7	1 799
Total	7	4 224 487

Table IV. **SE-CNN architecture.** The layers are listed in order, so the output size of the layer above will be the input size of the layer below. The first input size is $3 \times 100 \times 100$, corresponding to three color channels and 100×100 pixels in each image. The brackets in the convolutional (conv) and pooling layers contain [kernel/stride/padding]. The number in parenthesis in the SE layers is number of squeeze channels.

Layer	Output size	Trainable parameters
Conv [10/3/4]	$32 \times 33 \times 33$	9 632
SE (2)	$32 \times 33 \times 33$	162
Conv [3/1/1]	$64 \times 33 \times 33$	18 496
SE (4)	$64 \times 33 \times 33$	580
Avg pooling [2/2/0]	$64 \times 16 \times 16$	0
Fully connected	256	4 194 560
Output	7	1 799
Total	7	4 225 229

ReLU activation, except from output layers, which apply Softmax. Dropout is employed in all hidden fully connected layers, except from the second hidden layer of the MLP, where it was inadvertently omitted during implementation. We expect this to not be an issue, as a majority of the trainable parameters are found in the first layer (approximately 4×10^6 vs 3×10^5). This may result in a higher optimal dropout rate in the MLP to achieve strong enough regularization.

The neural networks are trained with backpropagation for eight epochs using a minibatch size of 128 samples. An epoch is defined here as one cycle though every sample of the dataset, meaning that for a training set of 12 271 samples, there will be $\lceil 12\,271/128 \rceil = 96$ parameter updates per epoch. Both choices were made after some preliminary trial and error, showing a balance between adequate convergence and training time. We use the Adam [10] optimizing algorithm for parameter updates. The three associated hyperparameters, β_1, β_2 and η, are tuned in a 3D grid search, holding the dropout rate constant at 0.5. Subsequently, dropout rate is tuned using the best performing Adam hyperparameters. Because of restraints on time available to run the grid search, we only do it once, in ranges chosen beforehand. We will therefore not have the opportunity to fine tune around the observed maxima. We do not tune hyperparameters related to the convolutional layers, such as stride or padding. Pooling layers passes on the average values, which has been done with good results in other computer vision tasks [11]. All weights and biases are initialized according to the He initialization scheme [12] (also known as Kaiming initialization). The weights and biases are sampled from a zero-centered normal distribution, with standard deviation $\sigma = 1/\sqrt{N_{\text{in}}}$, where N_{in} is the number of input values. This is shown mathematically and in practice to be well suited for networks using ReLU activations. We use the PyTorch [13] library for building and training the models.

G. Use of AI Tools

We have used AI tools for code debugging, in-code documentation and language refinement. We used OpenAI's ChatGPT [14] for debugging code, and to help improve the wording and readability of some sentences and paragraphs. The model was given prompts such as [give a suggestion for improving the clarity and flow of this sentence/paragraph], or parts of the code together with an error message. We used GitHub Copilot [15] to generate draft docstrings for functions, which we then reviewed to ensure correctness. Both ChatGPT and Copilot were used only as assisting tools, and all final code, results, analyses, and report writing are our own work.

III. RESULTS AND DISCUSSION

A. Hyperparameter tuning

1. *Solver parameters*

To identify the best performing Adam optimizer configurations, we performed a 3D grid search over β_1, β_2 and learning rate, η, for each model. The heatmap in Figure 6 shows validation accuracy for the SE-CNN as a function of β_1 and β_2, in the slice corresponding to the best performing learning rate, $\eta = 10^{-3}$. The highest validation accuracy is 73.7 % and is achieved at $(\beta_1, \beta_2) = (0.95, 0.99)$. The heatmap shows a broad plateau of high accuracies in the interval $\beta_1 \in [0.7, 0.9]$, $\beta_2 \in [0.8, 0.999]$, and the variation within this range may be due to noise from the stochastic initiation of weights and biases, and stochastic gradients. The noise from gradients could be mitigated by doing cross-validation in the grid search, but this was not feasible due to long training times.

Figure 7 shows the equivalent heatmap for the plain CNN. It seems to be more sensitive to the precise choice of β_1 and β_2 than the SE-CNN, showing a clear peak at

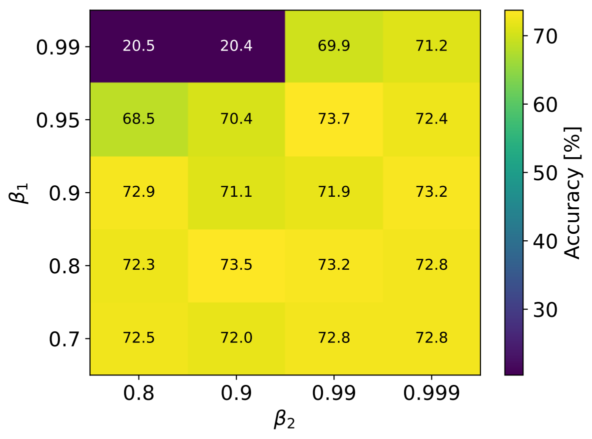


Figure 6. **Validation accuracy (%) from tuning of SE-CNN.** The heatmap shows performance across combinations of β_1 and β_2, evaluated at the best learning rate $\eta = 10^{-3}$. The highest validation accuracy of 73.7 % is achieved at $(\beta_1, \beta_2) = (0.95, 0.99)$.

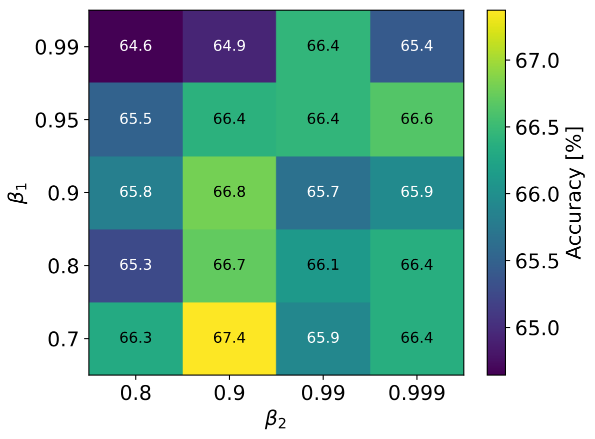


Figure 8. **Validation accuracy (%) from tuning of MLP.** The heatmap shows performance across combinations of β_1 and β_2, evaluated at the best learning rate $\eta = 10^{-4}$. The highest validation accuracy of 67.4 % is achieved at $(\beta_1, \beta_2) = (0.7, 0.9)$.

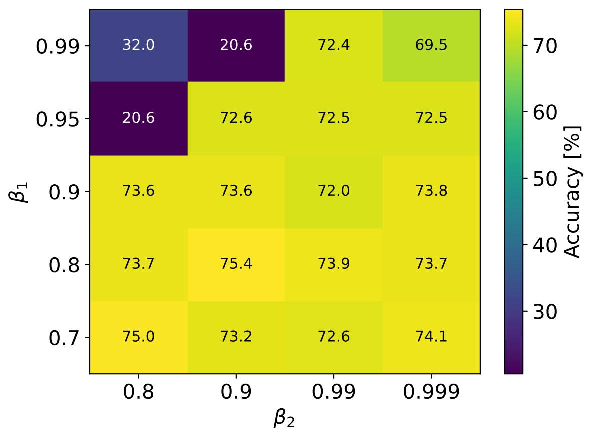


Figure 7. **Validation accuracy (%) from tuning of plain CNN.** The heatmap shows performance across combinations of β_1 and β_2, evaluated at the best learning rate $\eta = 10^{-3}$. The highest validation accuracy of 75.4 % is achieved at $(\beta_1, \beta_2) = (0.8, 0.9)$.

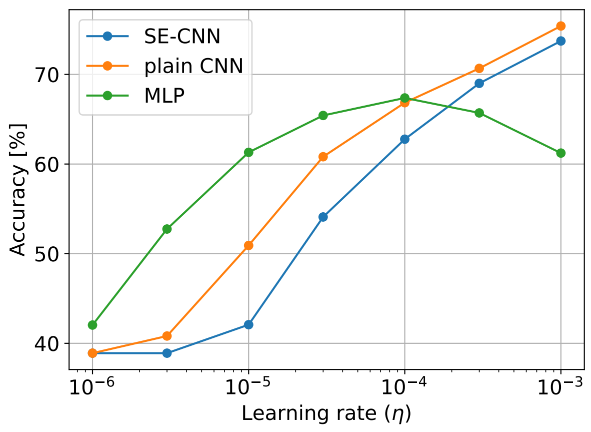


Figure 9. **Validation accuracy (%) versus learning rate η for all three models.** For each model, the parameters (β_1, β_2) are fixed at the best performing values. The highest validation accuracies are 73.7 % for SE-CNN and 75.4 % for plain CNN at $\eta = 10^{-3}$, and 67.4 % for MLP at $\eta = 10^{-4}$.

$(\beta_1, \beta_2) = (0.8, 0.9)$ with a validation accuracy of 75.4 %.

The heatmap for the MLP is shown in figure 8. It also has a clear peak, found at $(\beta_1, \beta_2) = (0.7, 0.9)$, where the validation accuracy is 67.4 %. The MLP seems to be more robust to suboptimal choices of β parameters than the CNNs, as it does not show the extreme lows observed in the upper-left corners of the CNN heatmaps.

Figure 9 shows validation accuracy with varying learning rate for all three models, evaluated at their respective optimal β_1 and β_2 values. The MLP has a clear maximum at $\eta = 10^{-4}$. For both CNNs, the highest validation accuracy is observed at the largest tested learning rate, $\eta = 10^{-3}$, indicating that the optimal learning rate may lie beyond the explored range. Based on the observed slopes, we cannot rule out that the SE-CNN could achieve a higher validation accuracy than the plain CNN if using the theoretically optimal solver parameters. A fine tuning in a range just above 10^{-3} would resolve this uncertainty.

2. *Dropout rate*

To study the effect of regularization, we tune the dropout rate for each model while keeping all other hyperparameters fixed. Figure 10 shows validation accuracy for all three models as a function of the dropout rate. The search was done with the best performing solver parameters. The values are quite noisy, especially for the CNNs, but clear trends are visible for all three architectures. For each model, validation accuracy increases with moderate dropout, indicating improved regularization and reduced overfitting. Beyond a certain point, however, higher dropout rates lead to a decline in performance, suggesting underfitting due to excessive removal of activations. The optimal dropout rates differ between models; the SE-CNN achieves its best validation accuracy at dropout rate 0.35, the plain CNN at 0.45 and the MLP at 0.30. We observe that the CNNs benefit from stronger regularization than the MLP.

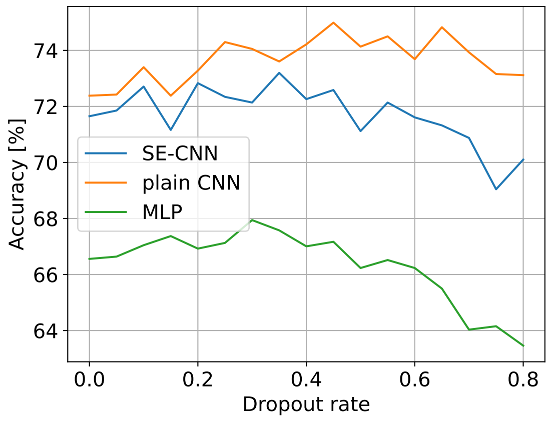


Figure 10. **Validation accuracy (%) versus dropout rate for all three models.** All other hyperparameters are fixed at their best performing values. The highest validation accuracies are 73.2 %, 75.0 %, 67.9 % for SE-CNN, plain CNN and MLP, respectively, and occur at dropout rates of 0.35, 0.45 and 0.30.

B. Test accuracies

The final hyperparameters and corresponding test accuracies for each model are summarized in Table V. The accuracies are consistent with the validation performance observed during hyperparameter tuning, indicating that the chosen configuration generalize well to the test set. Among the three architectures, the plain CNN achieves the highest test accuracy of 76.60 %, followed by the SE-CNN with 74.22 %, while the MLP achieves a substantially lower accuracy of 68.87 %. The results confirm the advantage of convolutional architectures for facial emotion recognition. The addition of squeeze-and-excitation blocks did not improve performance over the plain CNN in this examination. However, this comparison should be interpreted with caution, as the optimal learning rate for the CNN models may lie outside the range explored in the hyperparameter search.

To study how performance varies across classes, we analyze confusion matrices for the three models, shown in Figures 11, 12 and 13. The matrices are normalized along the true-label axis, such that each row sums to 1, and diagonal entries therefore represent the proportion of samples from a given true class that are correctly classified. Across all models, Happiness is consistently the easiest class to recognize, which is expected given that it is the largest class in the dataset (≈ 39% of training labels). In contrast, Fear and Disgust tend to have substantially lower recognition rate, and are frequently confused with other classes. This behavior is consistent with their limited representation in the dataset (Fear $\approx 2-3\,\%$, Disgust $\approx 5-6\,\%$). Across all models, we observe systematic misclassifications, most notably Fear being confused with Surprise or Happiness and Disgust being confused with Happiness or Neutral. Some of these confusions may reflect visual similarities between expressions, such as widened eyes and an open mouth in Fear and Surprise, or the relatively subtle visual differences between Disgust and Neutral. Fear or Disgust being predicted as Happiness is likely influenced by class imbalance, as Happiness is the most frequent class in the dataset. Overall, all models show a tendency to overpredict Happiness.

Table V. **Best performing hyperparameters and corresponding test accuracy for each model.** The values of β_1, β_2, learning rate η and dropout rate were selected based on validation accuracy.

Model	β_1	β_2	η	Dropout	Accuracy [%]
SE-CNN	0.95	0.99	10^{-3}	0.35	74.22
Plain CNN	0.80	0.90	10^{-3}	0.45	76.60
MLP	0.70	0.90	10^{-4}	0.30	68.87

Comparing the models, the plain CNN in Figure 12 shows the strongest diagonal entries overall and fewer pronounced off-diagonal confusions, in agreement with its superior test accuracy. It particularly improves recognition of Disgust compared to the other models. The SE-CNN in Figure 11 shows similar classification patterns to the plain CNN, but with lower overall accuracy. The MLP in Figure 13 achieves the lowest overall accuracy and shows more uneven performance across emotion classes.

One thing to note is that the accuracy is not a linear measure of how well the model is performing. For example, 38 % accuracy is easily obtained by always guessing Happiness. On the other hand, an increased performance on the minority classes, like Fear and Disgust, only gives a small increase in total accuracy. This is well demonstrated when comparing Figures 11 and 12. The biggest difference is the plain CNN correctly predicting Disgust twice as often (21 % vs 41 %), yet the total accuracy is only 2.38 percentage points higher. To better reflect performance across all emotions, we also report balanced accuracy, which weighs each class equally. This is defined as the mean of the diagonal entries of the row-normalized confusion matrix, and complements overall accuracy by giving equal weight to minority classes. The balanced

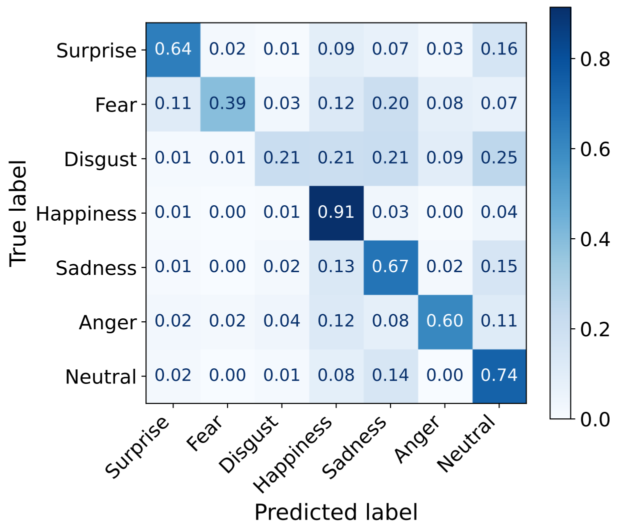


Figure 11. **Confusion matrix for SE-CNN evaluated on test set.** Rows correspond to true emotion labels and columns to predicted labels. The model achieves an overall test accuracy of 74.22 %.

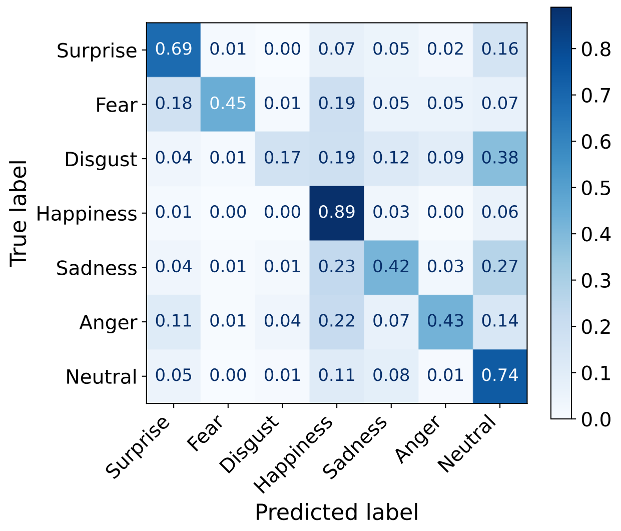


Figure 13. **Confusion matrix for MLP evaluated on test set.** Rows correspond to true emotion labels and columns to predicted labels. The model achieves an overall test accuracy of 68.87 %.

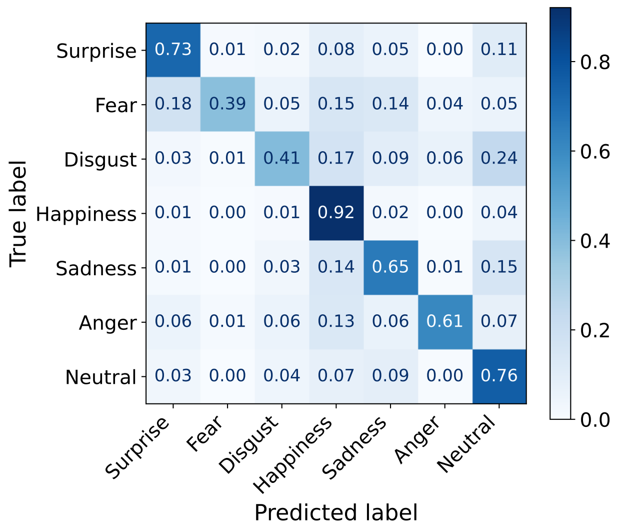


Figure 12. **Confusion matrix for plain CNN evaluated on test set.** Rows correspond to true emotion labels and columns to predicted labels. The model achieves the highest overall test accuracy of 76.60 %.

accuracy is then 63.86% for the plain CNN, 59.43% for the SE-CNN, and 54.14% for the MLP.

Huang *et al.* [2] trained their SE-ResNet, a modified version of the 18–layer ResNet with added SE blocks, on the RAF-DB dataset. The model contains approximately 11.27 million trainable parameters and achieves a reported validation accuracy of 65.67 %. All models considered in our work achieve higher test accuracies, at less than 40 % the size. While the MLP's achieves a comparable accuracy – close to the variation we observed due to stochastic initialization and training – the CNN models perform substantially better.

The reason for the comparatively lower performance of the SE-ResNet is not entirely clear. One possible explanation is the lack of hyperparameter tuning, which is not discussed in the paper and may indicate the use of default optimizer settings that are not tuned to the FER datasets. Our results also suggest that SE blocks provide little to no benefit, and may even been slightly detrimental. However, this is difficult to assess, as the architectures differ substantially; we use shallower but wider networks than the 18–layer ResNet. We also observe no signs of vanishing gradients in our networks, which is the primary motivation for introducing residuals [11]. Vanishing gradients would result in a lack of improvement during training and consequently low accuracy scores. This is mainly a problem with deeper networks, and our CNNs consist of only five layers (seven when including SE blocks).

A common challenge in facial expression recognition is demographic bias, where models perform well for certain groups but poorly for others. This issue is likely present in our models, as the RAF-DB is demographically imbalanced, with several ethnicities and age groups underrepresented. As a result, facial features associated with specific demographic groups may be spuriously correlated with specific emotions. For example, variations in skin color or facial structure could influence predictions if such features are unevenly distributed across emotion labels. A way to mitigate color information being used for predictions could be to apply random color filters during training to reduce the reliance on color-based cues. The best results would, however, be obtained by training on large datasets with broad demographic diversity. Future work could also evaluate model performance across demographic subgroups to quantify these biases.

IV. CONCLUSION

This study compares three neural network models in facial emotion recognition; an MLP, a CNN with SE-blocks, and a CNN without. The models are trained on the RAF-DB, a dataset containing 15 339 images of faces with emotion labels. We obtain the best results with the CNN without SE-blocks, which achieves a test accuracy of 76.60 %. The MLP and SE-CNN achieve test accuracies of 68.87 % and 74.22 %, respectively. We do not observe a clear performance improvement in the CNN when adding SE-blocks. However, this result is inconclusive because we did not fully optimize the learning rates. Both CNN architectures substantially outperform the MLP. This supports the standard practice of using CNNs for computer vision tasks. No signs of vanishing gradients are observed. All three models perform well compared to the SE-ResNet trained on the same dataset in the study by Huang *et al.* [2], which achieved a validation accuracy of 65.67 %. The lower performance of their larger network may be attributed to limited hyperparameter tuning. Our results suggest that careful optimization, particularly of the learning rate, can be more important than model size. Overall, this study highlights that smaller, well-tuned CNNs can outperform larger, less optimized models.

V. ACCESS TO DATASET

The dataset used in this work can be sent by the authors upon reasonable request. It can also be accessed by contacting the creators of the dataset[2].

[1] D. Mekinec, Facial emotion recognition: A complete guide (2024), blog post; published 2024-07-04, accessed 2025-11-17.
[2] Z.-Y. Huang, C.-C. Chiang, J.-H. Chen, Y.-C. Chen, H.-L. Chung, Y.-P. Cai, and H.-C. Hsu, A study on computer vision for facial emotion recognition, Sci. Rep. **13**, 8425 (2023).
[3] S. Li, W. Deng, and J. Du, Reliable crowdsourcing and deep locality-preserving learning for expression recognition in the wild, in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on* (IEEE, 2017) pp. 2584–2593.
[4] B. Bruno, C. S. Rønnestad, S. S. Svartbekk, and V. L. Marianayagam, Exploring Feedforward Neural Networks for Regression and Classification Problems (2025), unpublished report, University of Oslo.
[5] Wikipedia contributors, Convolution (2024), accessed: 2025-11-27.
[6] V. Dumoulin and F. Visin, A guide to convolution arithmetic for deep learning, arXiv preprint arXiv:1603.07285 (2016).
[7] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, Squeeze-and-excitation networks, IEEE Transactions on Pattern Analysis and Machine Intelligence **42**, 2011 (2019).
[8] T. Samavati, Squeeze-and-excitation block explained, `https://medium.com/@tahasamavati/squeeze-and-excitation-explained-387b5981f249`, accessed: 2025-12-10.
[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research **15**, 1929 (2014).
[10] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization (2017), arXiv:1412.6980 [cs.LG].
[11] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, CoRR **abs/1512.03385** (2015), 1512.03385.
[12] K. He, X. Zhang, S. Ren, and J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification (2015), arXiv:1502.01852 [cs.CV].
[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, PyTorch: An Imperative Style, High-Performance Deep Learning Library, in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)* (2019).
[14] OpenAI, Chatgpt (2025).
[15] GitHub, Github copilot (2025).

[2] `http://www.whdeng.cn/RAF/model1.html`